QGIS plugin para procesar datos geoespaciales en la nube

Raúl Estrada Herrerias

Resumen

Un grupo de investigadores ha desarrollado algoritmos de Machine Learning que procesan imágenes geoespaciales, estos procesos tienen costes computacionales considerables para ejecutarse en local. Para ejecutar estos datos existen algunas técnicas para procesarlos en máquinas externas. Este proyecto tiene como objetivo el desarrollo de un plugin de QGIS que sea capaz de procesar estas imágenes de remoto a un servidor en la nube ofreciendo diferentes servicios de procesamiento.
El servicio en la nube ejecutará los procesos en una máquina externa, tratará estos datos y enviará los resultados a QGIS para analizar esta información. Para obtener los resultados, los procesos se realizan en la parte del servidor montando un contenedor Docker con los puertos pertinentes, para así mediante la dirección de la máquina donde se monta el contenedor poder acceder a los servicios.

Palabras clave– QGIS, contenedor, Docker, usuario, servidor, nube, plugin, Python.

1 Introducción

QUANTUM GEOGRAPHIC INFORMATION SYSTEM [**1**] es una herramienta geográfica de código abierto que permite analizar y editar información espacial. Con ésta es posible editar, crear, visualizar y publicar cualquier información geoespacial en cualquier plataforma, como por ejemplo, Unix, Windows, Mac OS y Android.

Es posible acceder a una variedad de formatos y funciones, incluidos los basados en archivos (por ejemplo, archivos de forma ESRI, KML, GML), geodatabases (por ejemplo, PostgreSQL / PostGI, ODBC, ESRI Personal GeoDatabase, SQLlite) y protocolos de red (OPeNDAP, GeoJSON) [**2**] . Los datos están representados por dos tipos de capas nombrados a continuación:

- Las capas raster basadas en píxeles. Su información se organiza en una matriz 2D, donde cada celda de la matriz (píxel) proporciona datos de un área rectangular.
- Las capas vectoriales describen ubicaciones del mundo real utilizando puntos (un punto exacto), líneas (caminos) o polígonos (regiones).

QGIS [**3**] también ofrece herramientas muy potentes para así poder cargar, editar y visualizar información. No obstante la capacidad de estas herramientas para procesar imágenes está limitada, permite a los usuarios desarrollar sus propios procesos y scipts a través de un mecanismo de plugins. Como es posible observar en su repositorio [**https://plugins.qgis.org/plugins/**], contiene una gran variedad de plugins.

• E-mail: raulestra27@gmail.com
• Mención realitzada: Enginyeria del Software
• Trabajo autorizado por: Daniel Ponsa Mussarra (CVC)
• Curso 2020/21

2 Descripción del problema

Actualmente el grupo de investigación de MSIAU [**4**] utiliza QGIS para sus investigaciones. A medida que la tecnología avanza nos encontramos con más satélites, dando lugar a que estos entornos sean cada vez más potentes para descargar y visualizar imágenes utilizando QGIS. Los plugins que ofrece QGIS en la actualidad, nos ofrecen la posibilidad de utilizarlos para realizar operaciones especiales e interactuar con mapas. Estos plugins se instalan en QGIS y se ejecutan en la máquina siendo todo en local.

Una de las problemáticas con la que nos encontramos con el uso de estos plugins que QGIS ofrece, es el hecho de realizar procesos de manera local, puede ser muy costoso ya que se trata con grandes volúmenes de datos conllevando así, a procesar datos de gran tamaño con costes computacionales elevados, con ejecuciones lentas si no se dispone de hardware de procesamiento paralelo o muy potente.

Como consecuencia, si se crean plugins que ejecuten algoritmos en local puede conllevar a largas esperas para obtener resultados. Si necesitan operar o ejecutar una operación computacional que necesita muchos recursos, la máquina puede bloquearse al realizar esta operación puesto que se ejecuta en su propia máquina y esta puede estar limitada.

En este proyecto se propone desarrollar un plugin que gestione y procese los datos de QGIS en un servicio en la nube para poder así realizar operaciones computacionales de gran coste, procesando archivos de gran tamaño y mostrar los resultados deseados y sin necesidad de grandes recursos para la ejecución de este plugin remotamente.

Utilizando un servicio en la nube, las operaciones computacionales de éstos plugins pueden estar en el lado de un servidor, por lo que el grupo de investigación no necesitaría ejecutar estas operaciones en su propia máquina, proporcionando mayor velocidad y comodidad, sin necesidad de grandes recursos.

Centrándonos en el objetivo de este proyecto respecto a la implementación del plugin para QGIS, se ha de crear a un servicio en la nube para ejecutar operaciones computacionales con recursos externos figura 9, de este modo los investigadores utilizaran un plugin el cual se conectará con un servicio en la nube, con funciones integradas que les permita seleccionar múltiples opciones. Con tan solo el uso requerido de una interfaz sin la necesidad de otros plugins intermedios, con menos recursos y una ejecución mas rápida.

Utilizando un servicio en la nube [**5**] también se resuelve la problemática de limitación de una máquina, ejecutando y manipulando todo el proceso en un servicio externo (una máquina externa) con más recursos y mas potente, útil para mas de un usuario, pudiendo soportar múltiples conexiones y de esta manera solo se requiere un servidor potente.

3 Estado del Arte

Para aclarar el contexto de este proyecto, a continuación se explicarán algunos conceptos.

En la actualidad, los servicios en la nube se están expandiendo, el problema de las máquinas actuales son los recursos, dando lugar a recursos caros y limitados, el uso de la computación en la nube permite que un servicio en la nube utilice software de virtualización para hacer este proceso o guardar la información, sin necesidad de que el consumidor requiera de una máquina potente imagen [1].

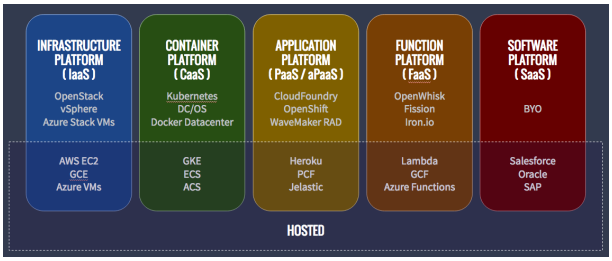


Fig. 1: Comparativa servicio en la nube

Respecto a los servicios en la nube podemos encontrarnos con 5 tipos:

- **IaaS** (Infraestructura como servicio). Un proveedor alquila una infraestructura y permite que el usuario controle toda esta infraestructura, como RAM, disco duro, etc. Este es el caso de plataformas como google cloud o amazon, que nos ofrecen todo lo que necesitemos pero a un coste que puede llegar a ser muy elevado.

- **PaaS** (plataforma como servicio). Ofrece una plataforma y un entorno para crear aplicaciones y servicios, estos servicios se pueden hacer con herramientas que ofrece el proveedor, por lo que estas herramientas están previamente configuradas, como una Operación del Sistema. Podemos encontrar plataformas GIS, que ofrecen el almacenamiento y computación de imágenes en la nube como:

  - *ArcGIS Server* [**21**]
  - *QGIS en la nube* [**23**]
  - *GIS en la nube* [**22**]

  Si que suelen ofrecer planes gratuitos pero cuando se necesitan para un uso profesional es necesario contratar un plan el cual puede ser muy costoso.

- **SaaS** (software como servicio). Permite a los usuarios el uso de una aplicación sin la previa instalación de un software. Este modelo en entorno GIS imagen [2] es compatible con otros modelos como:

  - *GaaS* (GIS como servicio).
  - *AaaS* (Aplicaciones como servicio).
  - *IaaS* (Imágenes como servicio).

  Ofreciendo ”servicios”que pueden contener una o mas tareas para el geoprocesamiento.

- **FaaS** (Funciones como servicio). Conocido también como arquitectura sin servidor, utilizando servidores como un elemento en la infraestructura. Se ejecutan como un contenedor efímero, es decir, podemos ejecutar o destruir el servicio en cualquier momento, así como, levantar varias instancias de un mismo contenedor, dicho contenedor se crea en el momento, de manera que el desarrollador no ha de gestionar la infraestructura sobre la que se ejecuta, centrándose, por tanto, en la funcionalidad, dando lugar a que cuando un contenedor se crea se consume y se destruye.

- **CaaS** (contenedor como servicio). Esto permite a los usuarios implementar y administrar aplicaciones mediante un método basado en contenedores utilizando centros de datos locales o la nube.

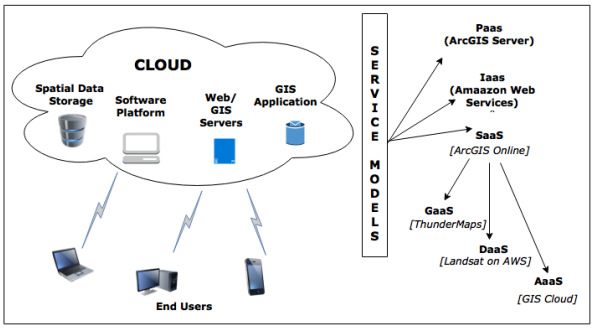


Fig. 2: Comparativa servicio en la nube

Una vez aclarado el concepto de servicio en la nube, nos centrándonos en **CaaS**, hay muchas aplicaciones para crear contenedores. Aplicaciones como es el caso de Docker, Kubernete, etc.

La idea de los contenedores es que tengan el mismo ciclo de vida que una aplicación y que sean efímeros, es decir, se puede ejecutar o destruir en cualquier momento y no solo esto sino que permiten una integración continua.

Un contenedor es una instancia de la imagen la cual se está ejecutando. Es posible tener tantas instancias de la imagen (contenedores) como queramos. En cierto modo, un contenedor consiste en una capa más sobre la imagen y existe sólo mientras se está ejecutando.

Por ejemplo, Docker [**6**] permite ganar escalabilidad y alta disponibilidad con sus contenedores, con Docker es posible crear contenedores para ejecutar una aplicación en cualquier máquina como un servicio, para usar en más máquinas y escalar recursos.

En caso de necesitar utilizar mas de una aplicación es posible incluso crear múltiples contenedores disponiendo en sí de un orquestador de estos, Docker Swarm [**19**].

Un ejemplo de cómo puede Docker ejecutar los contenedores imagen 3.

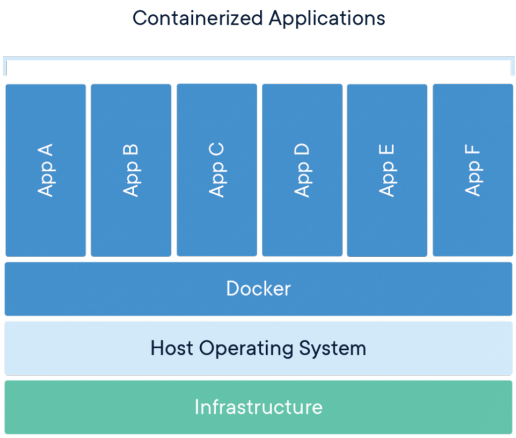


Fig. 3: Diagrama Docker

El sistema CaaS ha sido el elegido para este proyecto por ser un servicio totalmente gratuito y muy flexible, no obstante si es posible encontrar versiones de pago.

Docker es un servicio CaaS y opensource, al ser una herramienta popular hay mucha documentación y la curva de aprendizaje es rápida a diferencia de otras herramientas del mismo tipo.

Docker no solo ofrece esas ventajas sino que permite una instalación en cualquier máquina y con un fácil uso, permitiendo ejecutar contenedores con un gran abanico de lenguajes de programación, pudiendo ofrecer servicios.

4 Objetivos

El primer objetivo y enfoque principal es proporcionar un sistema o plataforma al grupo de investigación de MSIAU para que pueda aplicar fácilmente sus algoritmos de visión en ortomapas.

Para realizar este objetivo se propone, montar una aproximación a un servicio como CaaS en la nube para la ejecución de los algoritmos de visión en ortomapas.

Para realizar este servicio se utilizará la herramienta Docker[**7**], permitiendo crear un contenedor de una aplicación, para ganar escalabilidad y alta disponibilidad, ejecutando todas las operaciones computacionales en una máquina externa.

Este servicio será accesible a través de QGIS mediante un plugin, el cual se ha de desarrollar, para que sea capaz de conectarse a este servicio, dando lugar a funcionar como servicio en la nube, permitiendo a los usuarios utilizar este plugin sin consumir muchos recursos y procesar las operaciones con mapas en una máquina externa.

Para lograr todos los objetivos, este proyecto no comenzará como nuevo sino que continuará con su versión anterior con una codificación previa del plugin QGIS realizada por [**Oriol Casas**] `https://ddd.uab.cat/pub/tfg/2020/tfg_243876/Informe_Final.pdf`, para lograr los objetivos anteriores, como también los principales y no sobrescribir el código.

Para lograrlo, se deben alcanzar algunos objetivos específicos que se enumeran a continuación:

- Implementación de servicio en la nube (CaaS) que ejecute las operaciones algorítmicas de visión de computador.
- Implementación del proceso para incorporar los algoritmos de computación en el servicio nube.
- Codificar un plugin de QGIS para conectarse a todos los servicios disponibles en el servicio en la nube, interactuando y dando lugar a diferentes utilidades:
  - Consultar los servicios disponibles en el servicio nube y mostrarlo correctamente
  - Proceso para enviar una imagen, procesarla desde el servicio nube y seguidamente enviar el resultado a QGIS
  - Proceso para enviar una imagen de gran tamaño, recortarla en diferentes porciones para procesarlas por separado, juntar las porciones para obtener la imagen original procesada con el resultado deseado.

5 Metodología

Para lograr todos los objetivos, se dividió en tres fases para hacer una progresión continua.

La primera fase se centró en la recopilación de información y requisitos para el producto final.

Al mismo tiempo, una investigación previa sobre varios aspectos se ha realizado:

- Estudio sobre QGIS y la utilización de plugins, la forma en que trabaja, cómo desarrollar un plugin y como utilizarlo.
- Estudio sobre aplicaciones y plataformas para la realización de un servicio en la nube
- Estudio sobre contenedor como aplicación, en este caso Docker.

El Producto Mínimo Viable es el producto que tiene las características mínimas implementadas para considerarlo valioso para el usuario. El MVP se utiliza como referencia que demuestra que el producto se puede desarrollar y sienta las bases para su desarrollo.

Una vez el MVP quedó definido. La planificación se realizó en esta etapa, aunque si ha sufrido revisiones continuas por el flujo de trabajo iterativo.

Respecto a la segunda y tercera fase, son aquellas en las que el plugin y el servicio en la nube se han ido implementando. La segunda estaba más centrada en el desarrollo del MVP, para así tener una función mínima y conectarse al servicio en la nube.

La tercera se planteó como una fase para acabar con el MVP y completar el producto.

El plugin QGIS y el servicio en la nube tienen su propio repositorio creado en Github `https://github.com/LaidAkechi/TFG_QGIS_Cloud_Service`.

Los cambios de cada iteración han sido posteriormente subidos. De esta forma, si el desarrollo de la siguiente característica rompe el software, los cambios pueden ser invertidos fácilmente.

Para el desarrollo del servicio en la nube se ha utilizado Docker, seleccionado por su fácil aprendizaje y extensa documentación.

Las fases propuestas se planificaron utilizando el MS Project, realizando un diagrama de Gant fig.10 para ser persistente en los objetivos y lograr la finalización de las tres fases descritas.

5.1. MVP - Servicio proceso imágenes en la nube

El servicio en la nube se ha de ejecutarse usando una aplicación para que permita el uso de contenedores (como Docker) [**13**] y ejecutar el código en una máquina externa, para poder acceder desde cualquier dispositivo al servicio nube.

El servicio debe estar en funcionamiento y aceptar conexiones respondiéndoles. Este servicio debe poder administrar archivos JSON como peticiones y procesar información. Siendo todo ejecutado en la máquina del contenedor y almacenando todo en esa máquina.

Aparte de devolver su disponibilidad, el servidor debe responder a solicitudes que preguntan qué procesos y capas están disponibles.

Para el MVP, se plantean ofrecer 4 procesos disponibles:

- Guardar una capa/imagen en el servidor nube
- Cargar una capa/imagen previamente almacenada en el servidor nube.
- Cambiar la capa de colores de una imagen.
- Separar una imagen en varias porciones, operar con estas y devolver el resultado

Mediante estos sencillos procesos, se pretende evaluar la viabilidad de la solución propuesta. Finalmente, el servicio en la nube debe poder construirse y lanzarse como un contenedor Docker en una máquina para poder acceder a ésta externamente, como se aprecia en la imagen [4].

5.2. MVP - Plugin QGIS para acceder al servicio nube

El plugin debería poder conectarse a la nube y verificar si está disponible. Es el responsable de establecer la conexión y mostrar toda la información que necesita el usuario. Esta información incluye los procesos disponibles que ofrece el servicio nube.

El plugin guía al usuario mediante menús desplegables disponibles, al seleccionar un proceso este deberá ejecutar la operación que corresponde. También debe dar al usuario la posibilidad de abortar el proceso actual y mostrar en qué estado se encuentra.

Una vez que hay una salida o resultado, el proceso debe ejecutarse en la máquina externa y los resultados deben visualizarse en la máquina del usuario. Estos procesos se pueden hacer ejecutando este plugin en múltiples máquinas al mismo tiempo y con archivos pesados.

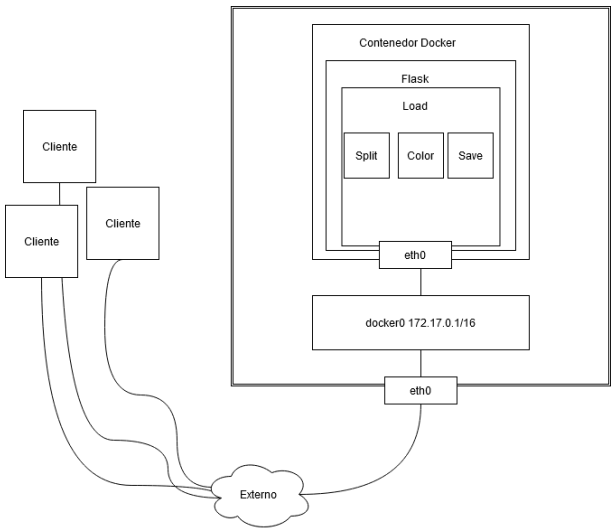


Fig. 4: Comparativa servicio en la nube

En caso de seleccionar cambiar capa de colores el servidor deberá devolver la imagen procesada sin el uso de archivos intermedios por la parte del usuario.

En el caso de seleccionar separar imagen, el servidor deberá cargar la imagen seleccionada por el usuario separar esta imagen en trozos en la banda del servidor, para su uso posterior y unir estos trozos para que el usuario reciba la imagen original procesada.

Las pruebas se realizarán en conexión local, dejando el contenedor en una máquina y conectado el plugin con otra de la misma red.

5.3. MVP – Refactorización e Implementación Final

Una vez que el contenedor y el plugin de Docker estén listos, éstos deben interactuar con las características descritas, se realizará una refactorización del código para permitir una fácil adaptación en características futuras.

Realizando así un manual con documentación para su fácil interpretación.

El servicio en la nube ha de estar en marcha en una máquina externa para su uso desde cualquier dispositivo mediante el uso del plugin de QGIS.

Se integrará en el servidor externo proporcionado y se comprobará que es capaz de realizar las funciones deseadas.

5.4. Herramientas utilizadas

Todo el código se cargará mediante la herramienta GitHub para mantener una versión de control del código [**9**].

Dado que QGIS utiliza Python [**10**] el uso de éste sera requerido.

Docker es necesario para los contenedores utilizando en sí Dockerfile, los requisitos son comandos y en el caso de Docker-compose yaml (este ultimo es útil para realizar mas de un contenedor al mismo tiempo, con interacción entre ambos).

Para implementar toda la lógica (codigo) como Python, Docker Y JSON, se ha utilizado la herramienta de Visual Studio. Siendo una potente IDE que ofrece la posibilidad de ser utilizada para múltiples lenguajes de programación y formatos en una sola herramienta.

Para utilizar los complementos de QGIS, se utiliza el software QGIS junto con su sistema de plugins para ejecutar los complementos.

Algunos módulos de Python són utilizados para realizar tareas especiales como Flask [**11**][**18**] o Opencv [**12**] para interactuar con imágenes.

Además también será requerido el uso de una máquina para ejecutar contenedores Docker y exponerlos.

Por último, para redactar toda la documentación se utiliza látex para que quede más claro y entendible, así como también herramientas para hacer una planificación, MSproyect y DIA para realizar diagramas.

6 RESULTADOS

Esta sección revisa los componentes del sistema que se han implementado y probado con éxito.

Siendo las funcionalidades del MVP los primeros elementos en ser terminados.

6.1. Requisitos e iteración de diseño

En la primera iteración, el propósito era definir el producto, comprender cuáles eran los propósitos y buscar una opción para diseñar un producto que se adapte a las necesidades de las partes interesadas, pudiendo así ejecutar las funciones de un plugin QGIS como servicio en la nube.

Los primeros pasos fueron encontrar un sistema o manera de crear una aplicación como contenedor, buscando información de las opciones existentes y metodología ha utilizar para esta.

También la realización de los diagramas y se redactaron especificaciones en esta fase. Mediante reuniones con el profesor se debatió varias veces la planificación y cómo el servicio nube sería más fácil de usar, qué características y metodologías son útiles y cuáles no.

De esta manera, pudiendo dar lugar a la idea de utilizar la aplicación como contenedor, intercambiando algunas ideas. Después de esta primera entrevista, las primeras pruebas fueron realizadas y discutidas en una segunda entrevista.

Para tener mejor comprensión del sistema y de como realizar los objetivos se realizo un árbol de extremos medios fig. 9. De esta manera, el plugin solo debe hacer peticiones y recibir las respuestas.

El servidor es responsable de supervisar el proceso. En la parte del servidor, todo el flujo de trabajo se realiza en esta banda.

Al diseñar una estrategia adecuada para ejecutar el plugin como un servicio en la nube, se evaluó como solución la utilización de Docker para ejecutar todo el proceso del plugin en una máquina externa como servicio en la nube (CaaS), una vez hecho esto, todos los procesos implementados deben mostrarse cuando se ejecuta este plugin.

Los contenedores corren en la máquina en entornos separados donde las aplicaciones se ejecutan sin compartir cualquier información con otros contenedores. De esta manera, se pueden hacer múltiples peticiones y no se superpondrán datos de otros procesos. Por ejemplo, App A no tiene información ni acceso a la Aplicación B. Estos hechos hacen de Docker un servicio ideal para implementar un servicio en la nube.

El sistema operativo no se especifica, ya que puede ejecutarse en varios sistemas operativos. Encima del SO, Docker monta sus contenedores.

Esto se hace según la planificación. Si que hay que mencionar que se sufre algunos problemas, como pasar el contenedor a un servidor externo o refactorizar el código para que funcione en todos los dispositivos, se pueden encontrar los problemas en la sección 6.6.

No obstante, como se describió anteriormente para la creación del contenedor se utilizó Docker, pero para la realización de la aparte lógica el uso de Flask es necesario para realizar la comunicación con los servicios que queremos ofrecer, siendo Flask una librería de Python que permite hacer micro servicios.

Para este proyecto, se realizó un Dockerfile (Lista código 1) para encapsular toda la lógica. Conteniendo los pasos que Docker debe seguir para construir las imágenes y qué comandos se deben ejecutar una vez que el contenedor esté funcionando.

Para construir un servicio en la nube, necesitamos que la máquina donde se ejecutará el contenedor tenga Docker previamente instalado para poder hacer uso de éste.

Se ha ejecutado un contenedor Docker de forma local y externa. Para comprobar las funcionalidades básicas de MVP como guardar o cargar capas, con diferentes tamaños. De esta manera, teníamos los conceptos básicos de las comunicaciones entre el complemento y el servicio en la nube.

Para un uso mas flexible el uso de Docker-compose[**17**] se ha implementado también para poder ejecutar el contenedor de manera mas automática y en un futuro si es necesario implementar varios contenedores (si es necesario comunicar con una base de datos externa), estos archivos utilizan el lenguaje yml siendo fácil de entender y extensible.

6.2. Implementación y pruebas

Mediante las pruebas realizadas para probar y actualizar el servicio en la nube, se ha ejecutado con más de una máquina al mismo tiempo y utilizando archivos grandes y pesados, para poder comprobar que se obtienen los resultados deseados.

Cuando un archivo es guardado, se asigna una marca de tiempo al nombre para evitar una sobre escritura si tiene otros cambios.

Además de realizar la primera implementación del servicio en la nube, también se añadió una opción para convertir una imagen, por ejemplo convertirla a GREY, para mostrar los resultados en local sin necesidad de recursos en local. Esta función guarda la imagen en el servidor externo la procesa y la devuelve procesada para que el usuario pueda ver los cambios realizados.

También, poder dividir una imagen en segmentos, ejecutado vía servidor, para así poder tratar con estos segmentos y procesarlos. Posteriormente enviar la imagen procesada y unida al usuario para mostrar el resultado en la máquina local.

Todas las funcionalidades se han probado ejecutándose en una máquina y dando múltiples peticiones.

Esto es parcial, por lo que la planificación sufre un retraso, hay algunos problemas con las librerías de Python cuando se intenta usar en un contenedor 6.6, estos problemas están resueltos.

6.3. Conexión del plugin de QGIS con el servidor en la nube

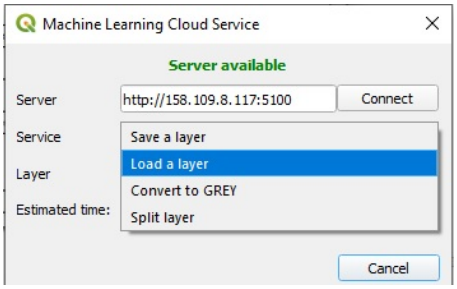
Para utilizar el plugin primero se debe cargar QGIS para utilizar su uso, cuando el plugin es ejecutado se puede acceder a la IP del servidor.

En la máquina externa, utilizada como servidor, se crea y se expone, utilizando el puerto pertinente para su uso, para que así el contenedor sea visible y accesible desde fuera. Al mismo tiempo este puerto tiene que ser el mismo que Flask ofrece, para poder seguir la comunicación entre ellos.

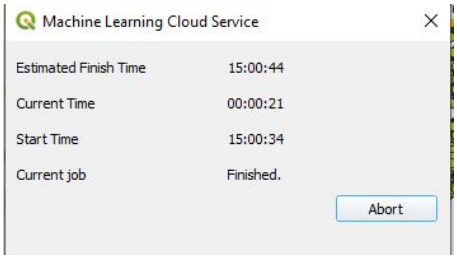
Para usar un contenedor y procesar capas, se requiere un contenedor con Python y la librería Opencv, por lo que se usa un contenedor con Opencv y Python previamente instalados para no tener problemas.

Una vez tenemos el contenedor montado y funcionando, ejecutamos el plugin de QGIS con la IP correspondiente podemos obtener los servicios como se puede ver en las imágenes fig.(5):

Fig. 5: QGIS plugin main menu



(a) QGIS plugin menu



(b) QGIS Plugin main dialog

QGIS solicita obtener una dirección utilizando el formato ”http://”seguido de la dirección donde se encuentra el servidor y el puerto al que apunta, cuando se asigna esta dirección.

Gracias a que los contenedores se ejecutan en esta máquina, permite que el contenedor puede ser utilizado por diferentes usuarios en entornos separados donde las aplicaciones se ejecutan sin compartir ninguna información con otros contenedores.

De esta manera, se realiza una prueba para ejecutar múltiples peticiones y no se superpondrán con los datos de otros procesos. Se utiliza para aislar diferentes procesos y aplicaciones.

6.4. Listado de servicios

Una vez accedemos al servidor y obtenemos el listado de servicios disponibles, podremos seleccionar que servicio queremos.

En el lado del servidor, el servicio en la nube 6 debería poder recuperar cualquier solicitud y procesarla.

Para probar la conectividad, la primera función a desarrollar estuvo relacionada con solicite su disponibilidad. El servicio devuelve HTTP 200 OK respuesta con el contenido de texto ”Disponible”.

Esta respuesta se hizo de esta manera, por lo que cualquier solicitud a esta función podría saber que el servidor es accesible, la respuesta 200, indicando que el servicio se llamó correctamente.

Una vez que un contenedor Docker está montado en la máquina, también es posible enviar múltiples conexiones al mismo tiempo como se puede apreciar en la imagen fig. 6

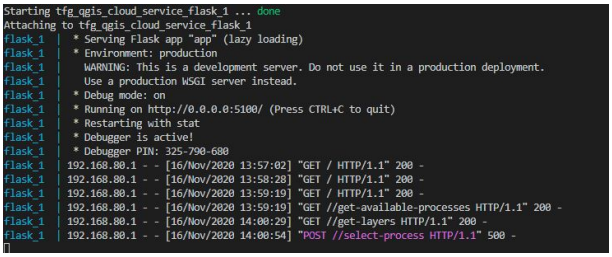


Fig. 6: Container debug messages

Cuando observamos dentro del log del contenedor podremos ver también el tipo de error puesto que muestra la opción seleccionada, también en el caso de cada función va mostrando mensajes de que capa esta tratando, para comprobar en caso de error en que paso se encuentra.

El siguiente paso a hacer en el servidor fue comprobar que los procesos están disponibles.

Para gestionar estos procesos, un archivo JSON se creó con todos los procesos disponible e información sobre ellos, así como una identificación para cada uno.

Para cada proceso, se creó una carpeta para diferenciar de las operaciones realizadas, ya sea para procesar, guardar, cargar o separar.

Además de los archivos de Docker, un script en Python se creó con el flujo que debe seguir cada proceso.

Este script se inicia una vez que el contenedor se está ejecutando.

El servicio en la nube procesa la solicitud y obtiene toda la información necesaria, qué proceso debe iniciarse y todos los parámetros necesarios para ejecutar.

También es responsable de guardar y administrar los archivos involucrados en el proceso. Un flujo de comunicación entre el servicio en la nube y el contenedor, está hecho para comunicarse entre ellos.

6.4.1. Descargar capas del servidor

Si el proceso seleccionado es çargar una capa ”, el servicio deberá devolver las capas disponibles.

No obstante para poder cargar una capa, previamente se han de dejar estas capas accesibles desde el contenedor.

Mediante un archivo JSON creado previamente con las capas y su información, se indicará la información de estas

capas y deberá contener los valores de estas capas para su posterior extracción.

Por tanto el archivo JSON deben ser editado manualmente.

Una vez tenemos capas en nuestro contenedor y esta opción es seleccionada, se nos mostrarán las capas disponibles para cargar en nuestra máquina local.

Seleccionada la capa a cargar, procedemos aceptar para cargarla y poder visualizarla en QGIS.

6.4.2. Subir capas del servidor

Cuando la opción Guardar imagen es seleccionada, podremos guardar cualquier tipo de capa. Una vez esta opción es marcada deberemos indicar la capa que queremos guardar, posteriormente se debe esperar hasta que se guarde y luego se podrá visualizar en el contenedor.

En este caso el archivo JSON mencionado anteriormente se edita automáticamente. Cuando una capa es guardada, el proceso se encarga de ir actualizando el contenido.

6.4.3. Aplicar proceso a una capa raster

Si la opción seleccionada es çonvertir a GREY", nos dará la opción de seleccionar que imagen deseamos procesar, una vez seleccionada el servidor cargará esta imagen, la procesara y nos la devolverá procesada, en este caso será convertida en GREY.

La opción de modificar los colores se edita manualmente y se pueden asignar cualquier tasa de colores con las convenciones que ofrece Opencv [**20**]. La modificación del script puede cambiar si se desea guardar la imagen en RGB, Black, etc., utilizando librerías de Python.

De esta forma, el proceso en el contenedor tiene todos los parámetros necesarios para ejecutarse y como podemos ver en las dos imágenes, no hay archivos intermedios, la imagen original 7 y otra en color GREY 8.

Fig. 7: Imagen original

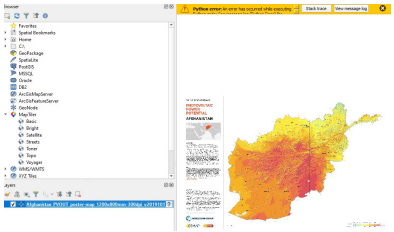
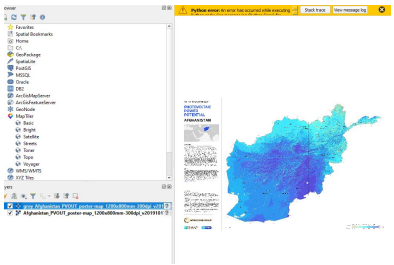


Fig. 8: Imagen convertida a Grey obtenida



Si queremos cambiar el tipo de procesamiento esto es posible editando manualmente el proceso encargado de realizar esta operación.

6.4.4. Aplicar proceso a una capa raster, con mosaico automatizado

Por último cuando la opción seleccionada es separar capa, deberemos indicar que capa queremos procesar, posteriormente el servidor cargará esta capa y la separará en trozos (modificable el número de trozos por código), los trozos de la capa pueden ser procesador desde el contenedor previamente a la unión ya que, se crea una carpeta "temporalçon el nombre del archivo a separar y sus partes para poder procesarlas.

Seguidamente se leerá esta carpeta para recomponer la capa y guardarla en una carpeta llamada "split", borrando la carpeta anterior de los archivos separados.

6.5. Pruebas finales

Una vez el servidor es capaz de devolver la petición seleccionada correctamente sin necesidad de una máquina potente ni archivos intermedios por la parte del usuario.

Se comprueba que el contenedor se puede ejecutar en el servidor proporcionado para así poder ser utilizado por MSIAU.

De este modo el servicio en la nube estará implementado y funcionando en el servidor de MSIAU. Realizando pruebas con múltiples conexiones y devuelve los resultados esperados.

Finalmente, se ha configurado el código para que sea fácilmente modificable para añadir funcionalidades nuevas y así su integración continua.

Así como la creación de un tutorial y documentación para entender el funcionamiento y modificación de éste.

También cuenta con mecanismos de seguridad y funciones relacionadas con tratar los archivos recibidos para evitar extensiones de riesgo para ser abierto o ejecutado en el servidor. Falta el servicio de un administrador para que nadie pueda gestionar los procesos JSON y las carpetas creadas para almacenar su información.

6.6. Problemas durante el desarrollo

Se expondrán los problemas que ha surgido en el desarrollo, dando lugar a algún retraso o cambios en la planificación como el foco de los objetivos.

El primer problema tubo lugar con el hecho de que el software QGIS era desconocido y nunca había trabajado con ello. Teniendo que aprender su funcionamiento, entenderlo y usar un plugin para ver su caso de uso.

También se han encontrado problemas a la hora de ejecutar el contenedor en diferentes arquitecturas de procesadores como el caso de ARM [**24**] (son procesadores utilizados en dispositivos pequeños que pueden llegar a tener bastante potencia), en las primeras versiones se utilizó una raspberry [**25**] con arquitectura ARM funcionando correctamente pero a medida que el proyecto avanzaba surgían problemas de compatibilidad de librerías Python, en su caso la imagen ya utilizada que incluye esta librería no es compatible con arquitecturas ARM, esto es debido a que la librería Opencv no da soporte completo para ARM y habría que buscar otras alternativas.

Al mismo tiempo, tenía que entender los propósitos del proyecto, lo que se debía hacer, no siendo fácil de entender

el funcionamiento, puesto que siempre puede haber mal entendimientos de que se espera del servidor para el usuario.

Respecto las tecnologías como Docker entenderlas a muy alto nivel y entender cómo aplicarlas correctamente, incluso saber exponer un servicio y hacerlo visible desde fuera.

A pesar de que estos puntos conllevaron a inseguridades en cómo avanzar en los primeros pasos y en cómo se deben desarrollar o comunicar el plugin y el servicio en la nube, a medida que encontraba más información y otras posibilidades, estas inseguridades desaparecieron.

Se pueden importar diferentes archivos con QGIS, por lo que las extensiones .shp crean algunos problemas que no permiten convertir el archivo o intentar leerlo, una vez que se comprende que son sólo las extensiones .tif las que se pueden editar. Esto se resolvió.

Las funciones más difíciles se han relacionado con la gestión de archivos y exportar diferentes capas siendo una tarea difícil, sin el conocimiento necesario sobre las librerías Python de QGIS, clases, métodos y aprendizaje profundo de contenedores Docker para comprender cómo funciona todo.

Librerías de Python como Opencv para poder tratar con imágenes y no solo como usar o entender la librería sino aplicarla a Docker puesto que esta librearía da problemas en algunos sitemas.

A la hora de entender cómo dividir archivos o hacer un cambio cuando el archivo se guarda con diferentes partes. Utilizando librerías como GDAL [**14**] tratando de dividir la imagen, se producen errores de Python que no se pueden resolver. Como alternativa para poder separar imágenes se utiliza una librería alternativa, image slicer[**16**], de fácil uso que comparte extensión con PIL para así soportar bastantes formatos de archivos.

Volviendo con las librerías como GDAL si ha sido probado pero sin éxito, estas librerías no son de fácil instalación y pueden conllevar a problemas con los contenedores. Así mismo se ha intentado obtener el CRS no solo con GDAL sino que también se intentado utilizar rioxarray [**27**] y rasterio [**28**].

El problema que conlleva no usar estas librerías, es que al tratar con imágenes geoespaciales no toda la información es guardada como el caso del CRS (coordinate reference system) [**26**].

Se utilizó Opencv para convertir las imágenes, el cual funciona en el lado del usuario perfectamente, pero una vez que esta librería se usó en el lado del servidor, hay algunos problemas que hacen que esta librería no funcione, usando una imagen de Docker con estas librerías previamente instaladas, este problema se resuelve [**15**]. A la hora de utilizar librerías para editar una imagen o capa, en este caso se ha utilizado Opencv, se encuentra el problema de que hay que utilizar una imagen con esta librería previamente instalada ya que instalándola directamente, hay problemas en el contenedor a la hora de su uso.

El hecho del gran desconocimiento entre la herramienta QGIS y otras como el caso de librerías Python y requisitos para crear el contenedor a ido atrasando un poco las iteraciones, fallos con las comunicaciones a la hora de crear contenedores y exponerlos, produciendo atrasos en el proyecto.

7 CONCLUSIONES

Gracias a la información recolectada para las especificaciones del sistema y estudiar las diferentes opciones que había, se ha podido ir avanzando progresivamente.

Se ha diseñado un servicio en la nube el cual permite hacer comunicaciones entre el usuario y una máquina externa, sin necesidad de que el usuario disponga de una máquina potente.

No obstante el MVP ha sido completamente desarrollado alcanzando los siguientes requisitos:

- Se ha configurado un servidor mediante Docker que sea capaz de utilizar Flask como aplicación para poder acceder externamente al servidor.
- Se ha conseguido que este servidor sea capaz de ofrecer a los usuarios diferentes servicios.
- Se ha podido hacer de tal manera que sea fácil hacer futuras modificaciones, para extender los servicios actuales o añadir futuros servicios.
- Se ha creado un tutorial para poder continuar con futuras revisiones.

Gracias a este proyecto se ha podido valorar la importancia que comporta crear un contenedor, para que realice operaciones complejas y poder utilizar los recursos de esa máquina sin necesidad de tener una máquina potente, ni recursos en nuestra propia máquina.

Por último se a podido comprender una pequeña parte del mundo de los servicios en la nube que es muy muy amplio, así como ver lo rápido que estas tecnologías avanzan para ofrecer servicios cada vez mas óptimos, potentes y rápidos.

7.1. Mejoras Futuras

El sistema desarrollado para este proyecto se puede extender y mejorar teniendo en cuenta varios puntos:

Se debe contemplar algún mecanismo de seguridad, el problema es que el contenedor esta expuesto hacia afuera y es posible atacarlo junto con su servidor.

Se ha utilizado en diferentes arquitecturas, pero en ARM podemos encontrarnos con problemas de compatibilidad con las librerías de Python puesto que librerías como Opencv no ofrecen una compatibilidad completa y la imagen utilizada para este proyecto de Python no es compatible con arquitecturas ARM.

Además, el servidor debe ser administrado por un administrador los JSON , caso de cargar una imagen se ha de hacer manualmente.

Hablando de tecnologías como Docker nacido en 2013, nos encontramos en la actualidad con una evolución continua de estas lo que supone cambios rápidos en las tecnologías, si miramos el futuro de Docker.

Docker es muy extenso tiene muchas opciones no seriamos capaces solo de cargar múltiples contenedores sino que también es posible ejecutarlos en diferentes máquinas pero el futuro de este no se ve tan claro pues hay herramientas mucho mas potentes que permiten mayor seguridad, mantenimiento y utilidad como el caso de Kubernetes.

Docker tiene una curva de aprendizaje corta y es lo que lo hace diferenciarse respecto a otros como el ejemplo de Kubernetes tiene una curva muy lenta pero ofrece muchísimas

mas opciones y mejoras, en cierto modo estudiar la alternativa de mirar hacia otra tecnología mas potente siempre es una buena alternativa pero se ha de estudiar el coste que supondrá.

Kubernetes a diferencia de Docker es mucho mas potente y ofrece mas alternativas a la hora de meter una aplicación en un contenedor.

8 AGRADECIMIENTOS

Agradecer a Daniel Ponsa por la ayuda prestada como tutor, guiando en el trabajo realizado y dando material de soporte.

Agradecer a los compañeros de ROCHE DIAGNOSTICS por la cantidad de material proporcionada como libros y ayuda sobre temas como Docker.

Agradecer a Oriol Casas por su proyecto previo realizado.

REFERENCIAS

[1] QGIS. visitada el 2020-09-15. 2020. URL: https://docs.qgis.org/3.10/es/docs/user_manual.

[2] Jae Wang -ISBN: 9781838644017 -Hands-On Geospatial Analysis with QGIS 3.10 and Python.-Released April 2020 -visitada el 2020-09-27

[3] Packt Publishing- -ISBN: 9781789341157 -QGIS Quick Start Guide. -Released January 2019 -visitada el 2020-10-05

[4] Universitat Autonoma Barcelona - Computer Center Vision visitada el 2020-09-21: http://www.cvc.uab.es/

[5] Wikipedia - servicio nube visitada el 2020-09-20: https://es.wikipedia.org/wiki/Computacin_en_la_nube.

[6] Docker. visitada el 2020-09-16. 2020. URL: https://docs.docker.com/.

[7] Sean Kane - Introduction to Docker Compose.(online training) -May 18, 2020 - visitada el 2020-10-27 URL: https://www.oreilly.com/live-training/courses/introduction-to-docker-compose/0636920433026/

[8] Docker. visitada el 2020-09-16. 2020. URL: https://docs.docker.com/.

[9] QGIS servicio nube -Raúl Estrada: https://github.com/LaidAkechi/TFG_QGIS_nube_Service

[10] Jae Wang -ISBN: 9781838644017 -Hands-On Geospatial Analysis with QGIS 3.10 and Python.-Released April 2020 -visitada el 2020-09-27

[11] Flask. visitada el 2020-09-17 URL: https://Flask.palletsprojects.com/en/1.1.x/api/

[12] Opencv Python - visitada el 2020-11-05: https://github.com/skvark/Opencv-Python

[13] -Tarek Ziadé -Packt Publishing -ISBN: 9781785881114 - Python Microservices Developmen. -Released July 2017 -visitada el 2020-10-30

[14] GDAL Python - visitada el 2020-11-05: https://gdal.org/

[15] jjanzic - Based on official Python:3 with addition of Opencv 3 visitada el 2020-11-20: https://samdobson.github.io/image_slicer/

[16] Image Slicer- visitada el 2021-1-5: https://github.com/janza/docker-Python3-Opencv

[17] Docker compose. visitada el 2020-10-12 URL: https://docs.docker.com/compose/

[18] Packt Publishing -ISBN: 9781784394783 - Flask Blueprints -Released November 2015 -visitada el 2020-11-01 URL:

[19] Mumshad Mannambeth. Docker -ISBN: 9781788991414 - Swarm, Services and Stack - Hands-On. -Released April 2018 -visitada el 2020-10-02. 2020.

[20] Opencv. documentación colores - visitada el 2021-1-5 URL: https://docs.Opencv.org/master/d8/d01/group__imgproc__color__conversions.html

[21] ArcGIS Server - visitada el 2021-1-20 URL: https://www.esri.com/en-us/arcgis/about-arcgis/overview

[22] QGIS en la nube - visitada el 2021-1-20 URL: https://support.qgisloude.com/kb/index.php

[23] GIS en la nube - visitada el 2021-1-20 URL: https://www.gisenlanube.com/about-us

[24] Arquitectura ARM - visitada el 2021-1-30 URL: https://www.arm.com/why-arm/architecture

[25] Raspberry - visitada el 2021-2-01 URL: https://www.raspberrypi.org/about/

[26] CRS - visitada el 2021-2-02 URL: https://cran.r-project.org/web/packages/eRTG3D/vignettes/v6.html

[27] rasterio - visitada el 2021-2-02 URL: https://rasterio.readthedocs.io/en/latest/installation.html

[28] rioxarray - visitada el 2021-2-02 URL: https://corteva.github.io/rioxarray/stable/

A APENDICE

El árbol de la Figura 9 muestra el árbol de extremos medios del proyecto. Este árbol se generó al comienzo de la elicitación para identificar los propósitos del proyecto en función de las necesidades que deben satisfacerse. En el recuadro central, encontramos el objetivo principal. Los recuadros superiores representan las necesidades que deben cumplirse y los inferiores representar los objetivos para satisfacer estas necesidades.

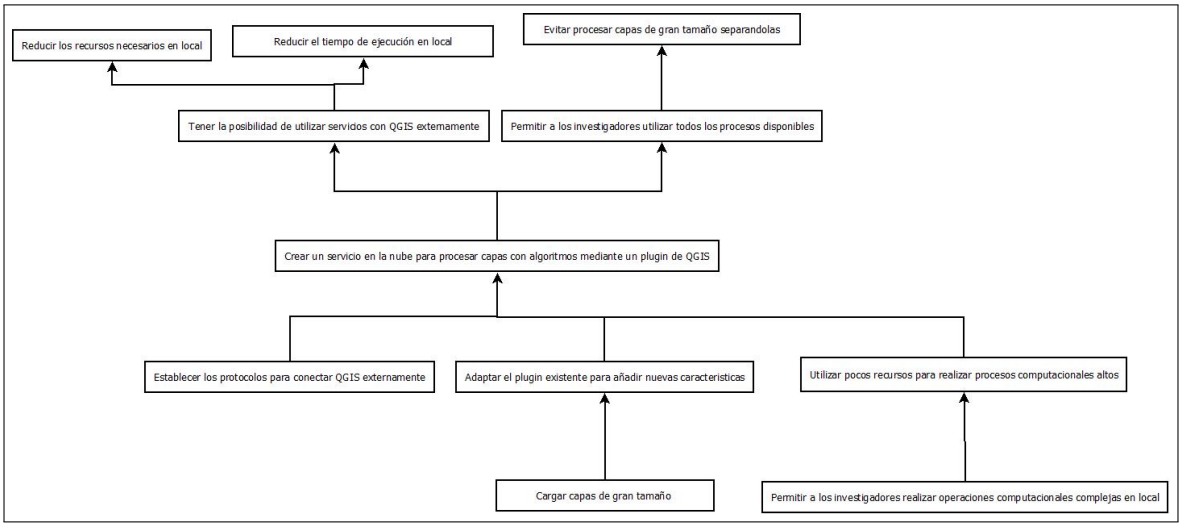


Fig. 9: Arbol extremos medios

A continuación encontramos el contenido de la Dockerfile, necesaria para poder ejecutar el servicio en la nube:

Listing 1: Dockerfile

```
# set base image (host OS)
FROM jjanzic/docker-python3-opencv

# We copy just the requirements.txt first to leverage Docker cache
VOLUME /app
ADD app /app
COPY ./requirements.txt /app/requirements.txt
WORKDIR /app
RUN python -m pip install --upgrade pip
RUN pip3 install -r requirements.txt
RUN pip3 install --upgrade setuptools
RUN pip3 install opencv-python numpy scipy
RUN pip3 install skia-python
CMD python3 app.py
```

Para aclarar todas las tareas, se hace un cronograma y diagrama de Gant con MSproyect, indicando como se ha ido realizando el proyecto.

Id	Nombre de tarea	Comienzo	Fin
1	**QGIS plugin para procesar datos georreferenciados en la nube**	**lun 14/09/20**	**jue 18/02/21**
2	**Definición de diagnóstico y MVP**	**lun 14/09/20**	**dom 11/10/20**
3	Buscar informacíon sobre como crear un sistema cloud para la comuniacion del plugin QGIS	lun 14/09/20	jue 17/09/20
4	Planificación	lun 14/09/20	mar 15/09/20
5	Informe inicial	lun 14/09/20	lun 14/09/20
6	Entrega informe inicial	jue 08/10/20	jue 08/10/20
7	**Primera implementación del MVP**	**dom 11/10/20**	**dom 15/11/20**
8	**MVP Servicio proceso imágenes en la nube**	**lun 12/10/20**	**vie 06/11/20**
9	Revisar los objetivos	lun 12/10/20	lun 12/10/20
10	Análisis de plugins / sistema QGIS actual	lun 12/10/20	vie 16/10/20
11	Evaluar los sistemas existentes sobre servicios en la nube	lun 19/10/20	mar 20/10/20
12	Implementación de una aplicacions como contenedor (CaaS)	lun 12/10/20	vie 16/10/20
13	Primera implementación de una aplicación como contenedor	vie 30/10/20	vie 06/11/20
14	**Primeros tests**	**lun 12/10/20**	**vie 16/10/20**
15	Creación / configuración de github y herramientas necesarias	lun 12/10/20	mar 13/10/20
16	Primeras pruebas para comprobar su correcto funcionamiento (exponer contenedor)	mié 14/10/20	vie 16/10/20
17	Realizar test con multiples conexiones y con imágenes de varios tamaños	lun 12/10/20	mar 13/10/20
18	**Documentación**	**lun 12/10/20**	**vie 13/11/20**
19	Sistema de especificación de documentos elegido	lun 12/10/20	mié 14/10/20
20	Informe previo a la entrega del primer informe	lun 09/11/20	lun 09/11/20
21	Primera entrega del Informe de progreso	vie 13/11/20	vie 13/11/20
22	**Segunda fase MVP**	**lun 16/11/20**	**dom 20/12/20**
23	**Plugin QGIS para acceder al servicio nube**	**lun 16/11/20**	**lun 30/11/20**
24	Implementación final del sistema para ejecutar todos los scripts en la nube	lun 16/11/20	lun 23/11/20
25	Listado de servicios funcionales (convertir y separar imágen)	mar 24/11/20	lun 30/11/20
26	**Fase de test**	**lun 16/11/20**	**mié 18/11/20**
27	Prueba de los cambios realizadas y los resultados esperados	lun 16/11/20	mié 18/11/20
28	**Documentación**	**lun 16/11/20**	**mié 18/11/20**
29	Extracción de resultados y conclusiones provisionales	lun 16/11/20	mar 17/11/20
30	Segunda entrega del informe de progreso	mié 18/11/20	mié 18/11/20
31	**Fase final**	**dom 20/12/20**	**jue 18/02/21**
32	**Refactorización e Implementación Final**	**dom 20/12/20**	**lun 04/01/21**
33	Integración del servicio cloud en el servidor proporcionado	dom 20/12/20	lun 21/12/20
34	Refactorizar el código para una fácil extensión	lun 21/12/20	mié 30/12/20
35	Pruebas finales y modificaciones	lun 21/12/20	lun 04/01/21
36	Evaluación de resultados esperados	lun 21/12/20	jue 24/12/20
37	**Documentación**	**lun 21/12/20**	**vie 08/01/21**
38	Redacción y entrega de la propuesta del informe final	lun 21/12/20	mié 30/12/20
39	Creación de tutorial para el plugin y servicio en la nube	jue 31/12/20	lun 04/01/21
40	Entrega final	mar 05/01/21	vie 08/01/21
41	**Ultimos detalles**	**vie 05/02/21**	**jue 18/02/21**
42	Propuesta incial de presentación	vie 05/02/21	dom 07/02/21
43	Defensa del TFG	jue 18/02/21	jue 18/02/21

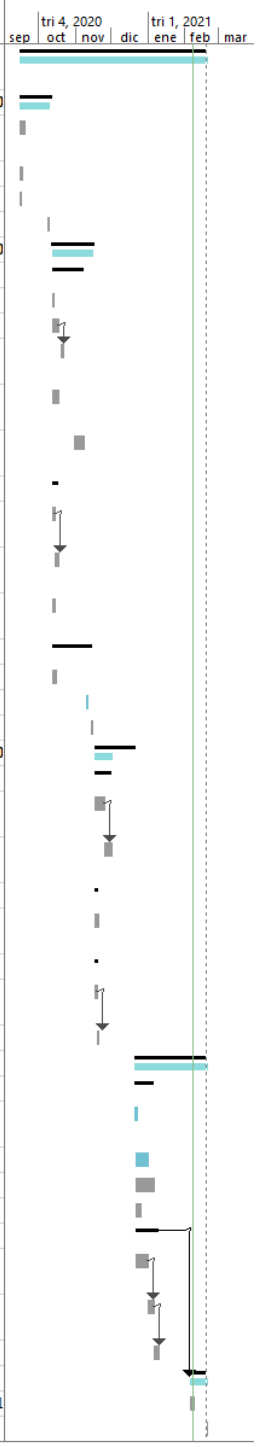


Fig. 10: Diagrama de Gant